Lessons Learned in Developing a Low-cost High Performance Medical Imaging Cluster

Kirt Lillywhite[a] Dah-Jye Lee[a] Sameer Antani[b] Dong Zhang[c]
Rodney Long[b]
[a] *Department of Electrical and Computer Eng., Brigham Young University, Provo, UT*
kirt.lillywhite@gmail.com, djlee@ee.byu.edu
[b] *National Library of Medicine, National Institutes of Health, Bethesda, Maryland*
antani@nlm.nih.gov, long@nlm.nih.gov
[c] *The School of Information Science and Technology, Sun Yat-sen University,*
Gaunzho, Guangdong, 510275, China
zhangd@mail.sysu.edu.cn

Abstract

This paper explores the usefulness of the Sony PlayStation 3®(PS3) for medical image processing. Medical image processing often entails dealing with a large number of high resolution images, requiring a large amount of computational power to process. The PS3 is powered by the Cell Broadband Engine, a microprocessor created by IBM, capable of rapid numeric computation with low power requirements that has helped the unit to become a popular gaming unit. The unit can be repurposed as a low-cost high performance computing platform. In order to demonstrate the computational abilities of the PS3, several basic image processing tasks are implemented and compared with desktop PCs, equipped with general-purpose microprocessors. This article describes lessons learned in the process of building some fundamental image processing tasks. The article also describes the architecture of the Cell Broadband Engine and provides information about developing applications given the architecture. The article also provides an introduction to setting up a high-performance image processing environment with a cluster of such relatively inexpensive PS3 units.

1 Introduction

Medical image processing tasks are often highly computationally intensive. Applications routinely process a large number of high resolution images. Both challenges can result in lengthy run-times and require appropriate capabilities. To address the challenges posed by such intensive computation this article explores the possibility of building a cluster of Sony PlayStation 3®(PS3) gaming consoles. The PS3 was developed for gaming but has several features that make it attractive for scientific computing: (i) the PS3 is designed to easily install other operating systems, such as, several Linux distributions, that support PowerPC processors; (ii) is relatively inexpensive; (iii) and it uses the Cell Broadband Engine (Cell BE) as its main processor. The PS3 is relatively inexpensive because gaming consoles are a very highly competitive market, with manufacturers often selling their units at a loss in order to make money later through games, accessories, etc.

The Cell BE, is a heterogeneous multi-core processor jointly developed by Sony, IBM, and Toshiba. The project grew out of a challenge from Sony and Toshiba to create a power efficient and cost effective high performance processor [1]. The Cell BE is designed to be used in a variety of applications, and especially those that require intense single precision computational abilities. The Cell BE is also used in the Roadrunner supercomputer at Los Alamos National Labs in New Mexico which in November 2008 was ranked number one on the Top500 list [2], which ranks supercomputers worldwide. As evidence of efforts to make the Cell BE power efficient, the top seven supercomputers on the 4th edition of the Green500 list [3] released in November of 2008 use the Cell BE. The Green500 list ranks supercomputers by performance per Watt. The Roadrunner supercomputer in Los Alamos National Labs is ranked

seventh on the Green500 list.

To further increase the performance of a single PS3, a cluster of PS3 units can be assembled. Clustering computers is an old concept and has been used frequently as a means to distribute computation [4]. Building clusters of PS3s [5, 6, 7] has become popular in recent years and such clusters are also available commercially.

There is a need for a freely distributed image processing library and better documentation of cluster setup procedures so that PS3 units can be more widely used in scientific applications. The rest of the paper is organized as follows: Section 2 discusses the architecture and programming model for the Cell BE. Section 3 discusses the programming environment for the PS3. In Section 4, two basic image processing tasks are implemented on the Cell BE and compared with other microprocessors. The cluster setup is discussed in Section 5. Finally, Section 6 lists our conclusions and direction for future work.

2 Cell Processor

The Cell BE is a heterogeneous processor comprising of a Power Processing Element (PPE) and eight SIMD[1], co-processors, called Synergistic Processor Elements (SPEs). The eight SPEs deliver most of the computational abilities of the Cell BE. The SPEs have been tuned for numerical computation and rely on a simple design with short pipelines and a SIMD instruction set. Single precision operations are fully pipelined within the SPEs, making each SPE capable of a peak performance of 25.6 Gflop/s. All together the SPEs are capable of 204.8 Gflops/s. Hardware controlled caches, dynamic branch prediction, out-of-order execution, and other space consuming hardware have been removed to make the SPEs simpler and faster. The SPEs have two execution pipelines whose selection for use is determined by the instruction type.

The processing cores are joined by a high-bandwidth interconnect bus, called the Element Interconnect BUS (EIB). The PPE is used mainly to handle the demands of the operating system and to control the eight SPEs. The PPE is a PowerPC core, similar to those that might be found in more traditional microprocessors. The PPE does not support out-of-order execution, but does have simultaneous multi-threading support, allowing two threads of execution at the same time. The PPE has 32 KB of L1 cache and 512 KB of L2 cache.

For data transfers each SPE has a memory flow controller (MFC) to manage DMA operations, allowing data transfers to overlap with computations. Each SPE has 256 KB of memory called the local store (LS). The SPEs only have access to their own LS and all data in RAM must first be transferred to it before use. The LS is not a traditional hardware cache and does not have mechanisms to predict memory accesses. It acts as a software cache that must be explicitly controlled by the user. This design allows greater and more predictable performance than traditional caches because application-level knowledge can be used to effectively pre-fetch all data that are to be used. This can be an effective method for minimizing memory latency [8]. Separate MFC in each SPE permits techniques such as double buffering or multi-buffering, to reduce or often times completely eliminate delays due to memory transfers. Figure 2 shows how double buffering can completely eliminate every memory transfer except the very first transfer if the computations involved take as long or longer than the DMA transfers. Figure 1 shows, at a high level, the architecture of the Cell BE.

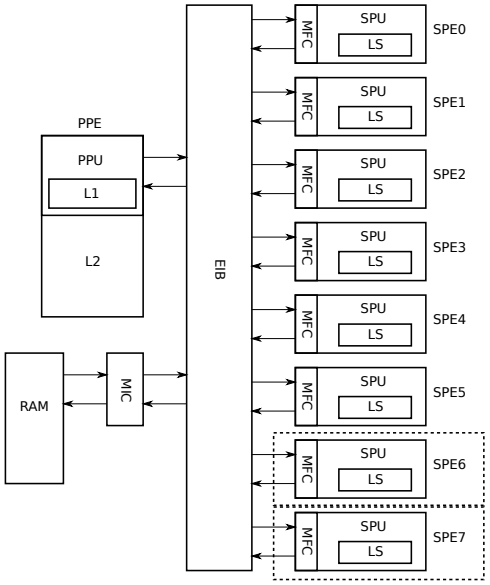

Figure 1. Diagram of the Cell BE [9]. On the PS3, one SPE, shown outlined with dashed lines, is reserved for the operating system and the other SPE outlined is disabled to increase manufacturing yields. This leaves six SPEs available to the system.

While using a software cache can lead to better memory performance, it does make the programming paradigm more difficult for the programmer. The 256 KB LS is used to store instructions as well as data and must be used carefully in order to not exceed its capac-

[1]Following Flynn's taxonomy SIMD stands for Single Instruction, Multiple Data streams.

ity. Additionally, DMA operations must follow a set of rules in order to not generate run-time errors.

- DMA operations must be 1, 2, 4, 8, 16, or any multiple of 16 bytes in size.
- For 1, 2, 4, or 8 byte transfers, the source and destination addresses must have the same four least significant bits.
- Data transfers must be less than 16 kilobytes in size.

In addition to rules to ensure DMA completion, there are other factors that can affect DMA transfer speed. If the source and destination do not have the right alignment it will require two DMA operations rather than one.

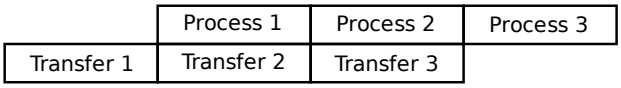

Figure 2. Processing cannot begin until the first data transfer has completed. As the first data transfer is being processed, the second data transfer can occur. If the processing takes long enough it can completely hide the second and subsequent data transfers.

Each core on the Cell BE also has mailboxes for sending 32 bit messages to each other. The mailboxes are simply small buffers for holding messages. These mailboxes are mostly intended for sending status messages but can also be used to send other short data transfers and typically have a lower latency than DMA operations.

One of the most significant challenges when working with the Cell BE is programming for multiple cores. While multiple core processors are quite common now, software developers are still slow to adopt new programming paradigms to take advantage of multiple cores. These machines, however, still benefit when multiple applications are run concurrently as each core can handle an application. The Cell BE though has heterogeneous cores and if individual applications do not utilize multiple cores then all applications will run on the one PPE core, making programming for multiple cores essential.

To use the multiple cores on the Cell BE for medical image processing, several parallel programming approaches can be used.

Pipeline The cores could be considered as a hardware pipeline in which each core has a specific task to complete and passes its output to the next core which is the next stage in the pipeline. Advantages to this approach are that SPE to SPE transfers are very fast and more than one task can be done at once, possibly reducing the amount of overhead due to moving code in and out of cores. The disadvantage to such an approach can be when the run-time of tasks are not balanced resulting in underutilization and inefficiency.

Data Partitioning Another very common way to use the cores in parallel is to break up the data into parts and to process them in parallel. Each core runs the same code but with different data. This has clear block separation where each SPE is given a piece of the image to operate on. One disadvantage, however, is if there are other tasks that need to be accomplished, the SPEs have to be stopped, new code loaded, and the SPEs started again.

Task Parallelism In a task parallel approach, each core runs a different task that is independent from the other tasks. If the tasks share data, considerable resources can be consumed to handle dependencies resulting in inefficiencies.

The Cell BE in the PS3 has some restrictions. Only 6 SPEs are available for use. One of the SPEs is reserved by the operating system and another has been disabled to allow higher manufacturing yields.

3 Programming Environment Setup

In order to use the PS3 for medical image processing several changes need to be made to the factory setting on the units. We chose Yellow Dog Linux 6.0, maintained by Fixstars[2]. Yellow Dog Linux is based off of Red Hat Linux and uses the familiar RPM package manager. This distribution of Linux is designed specifically for the PS3, and is relatively easy to install and use. It also contains the freely distributable parts of the IBM Cell SDK 3.0. After Yellow Dog was installed, then the remaining components of the Cell SDK were installed. The SDK provides useful libraries and the ability to use the SPEs.

Fixstars has a install guide on their website that can be followed to install Yellow Dog Linux on the PS3. In order to do the installation and use the Linux environment it is strongly recommended that a high definition LCD monitor or TV is used so that text can be read. A computer monitor can be used with the use of an HDMI to DVI adapter as long as the monitor has support for high-bandwidth digital content protection (HDCP).

[2]http://us.fixstars.com/products/ydl/

For compiling, the GNU compiler GCC was used. There are two separate GCC compilers that are needed, one for the PPE and one for the SPEs. The SPE code is compiled into a static library and linked into the PPU code at link time during the compilation.

In order to get accurate timing information, special hardware timing resources on the Cell BE are used. There are a total of 11 counting registers on the Cell BE that can be used for timing purposes; three on the PPE and one on each of the SPEs. The counters are incremented or decremented at the timebase frequency, which can be found in the `/proc/cpuinfo` file in a Linux environment. Each SPE has a decrementing 32 bit counter. The counter is written to its maximum value at the start of a timing block and then read at the end of the block. At the timebase frequency of the Cell BE found in the PS3 this counter will underflow at 53.82 seconds and have a resolution of 12.53 nanoseconds. On the PPE the timebase register is used directly. It is a 64 bit incrementing counter allowing to time longer running blocks; up to 231 billion seconds long. The IBM SDK provides intrinsics for working with the timebase register.

4 Example Medical Imaging Building Blocks

In order to show the abilities of the Cell BE, simple image processing functions that make up the medical imaging algorithm in our application, were programmed on the Cell BE. The first function converts an image from the RGB color space to the HSV color space[10]. It is simple but is an image processing function that used in many algorithms. The second function is a Gabor filter[11] and is a little more complex but also shows up frequently in image processing.

Since the PS3s will be placed in a cluster a data partitioning approach to parallelism has been taken. Each core on the Cell BE handles a single image, processing 6 images at a time.

4.1 RGB to HSV Conversion

On a normal PC, OpenCV[3] was used to do the RGB to HSV conversion. OpenCV is an open source image processing library first developed by Intel Corporation and optimized for Intel Pentium®processors. As of 2008 OpenCV is supported by Willow Garage, Inc. It is released under a very liberal BSD style license, which places no restrictions on the redistribution of the source code or binaries as long as it maintains the copyright. Due to the liberal license and optimizations made in OpenCV, it is a perfect starting place to build basic building blocks for a medical image processing library for the PS3.

The OpenCV code for RGB to HSV color conversion was ported to the Cell BE. The only modification needed to be made was in the manner in which the data was loaded. The code had to be wrapped with DMA operations to move the image data from the main memory to the LS of the SPEs and the results back to the PPE. So there was a very minimal amount of work that went into porting the code to the Cell BE.

Although porting the code was simple, there were some problems with getting desired performance. A good amount of time was spent trying to speedup DMA operations that were under-performing similarly implemented DMA benchmark programs. The way that the results were allocated in main memory proved to be very important. At first the results were allocated using a call to the function `malloc_align`, which works like `malloc` except the memory alignment is specified. The second time the results were allocated in the same way but then assigned values, by loading a blank image into the allocated memory. The second method led to considerable performance increase allowing six 1500×1024 images to go through color conversion in 42 milliseconds rather than 274 milliseconds. Table 1 shows how long the RGB to HSV color conversion took in comparison to other processors.

Processor	Time for RGB2HSV
PowerPC on PS3	445 ms
Pentium 4 (3.4 GHz)	160 ms
Phenom 2 quad-core (2.8 GHz)	71 ms
Cell BE	42 ms

Table 1. Comparison of four processors to do RGB to HSV conversion for six images that are 1500×1024 in size. The code is written to take advantage of multiple cores if they are present.

4.2 Gabor Filter

The Gabor filter is a bandpass filter, which has been shown to approximate the visual cortex in some mammals [12]. Gabor filters have been used for edge detection, texture segmentation, retina identification, document processing, and many other applications. In many applications a bank of Gabor filters is used with each filter having a different set of parameters. Gabor

[3] http://sourceforge.net/projects/opencvlibrary/

filters works like other linear image transforms in which the image is convoluted with a kernel. The Gabor filter kernel was created using Equation 1.

$$g(x, y; \lambda, \theta, \phi, \sigma, \gamma) = e^{-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}} \cos(2\pi \frac{x'}{\lambda} + \phi) \quad (1)$$
$$x' = x\cos\theta + y\sin\theta$$
$$y' = -x\sin\theta + y\cos\theta$$

In order to take advantage of the SIMD nature of the SPEs, it is best to operate on vectors rather than scalar data types. The IBM SDK provides intrinsics, which are C language instructions that essentially inline assembly instructions. The intrinsics allow developers to operate on vectors. The GCC compiler, however, can in many cases create SIMD code without the use of intrinsics through a process called *autoSIMDization*. Since there are a large number of image processing functions that will be implemented in the future, we try to avoid hand coded optimizations using intrinsics and rather let the compiler make the optimizations for us. The first Gabor filter implementation took 1000 milliseconds to do six 1024×1024 images, making the Cell BE implementation approximately 1.5× faster than the Pentium 4 implementation. Much of the success of autoSIMDization depends on how loops are written. The innermost loop of the Gabor filter was completely unrolled and allowed the compiler to make further optimizations. After loop unrolling, the function took 166 milliseconds to run, giving approximately a 9× speedup over the Pentium 4 implementation. Table 2 shows how long it took to do one Gabor filter on several processors.

Processor	Gabor Filter Time
PowerPC on PS3	19,580 ms
Pentium 4 (3.4 GHz)	1,510 ms
Phenom 2 quad-core (2.8 GHz)	322 ms
Cell BE	166 ms

Table 2. Comparison of four processors to run Gabor filter over six 1024×1024 images. The code is written to take advantage of multiple cores if they are present.

5 Cluster of PS3s

The main goal of this project is to enable processing a large number of high-resolution medical images. The images require different image processing operations to be performed on them. To do all this processing can take several weeks on multiple high-end PC workstations with methods implemented in MATLAB. We aim to build a cluster of PS3 machines that can process these images much faster at a low cost and complexity.

As shown in Figure 3, the cluster has one workstation that is used to store images and to control the work done on the cluster. The workstation and PS3s are connected to a high speed gigabit switch. The PS3 and workstation all have gigabit network interface cards (NIC). Following the advise given by Buttari et al. [7] Jumbo Frame support has not been enabled because of the configuration difficulties and limitations of the NIC in the PS3. Jumbo Frames allow a higher transfer bandwidth by increasing the size of Ethernet frames that travel across the network.

Each PS3 is headless with no graphical environment loaded and all unnecessary daemons are not loaded in order to conserve memory space and increase efficiency. Work is done on the PS3 using Secure Shell (SSH) from the workstation. Communication between nodes of the cluster, including the workstation, are done using MPI. The Open MPI [13] version 1.3 implementation of MPI is used, compiled for use in a heterogeneous environment. Open MPI is an open source implementation of both the MPI-1 and MPI-2 standards.

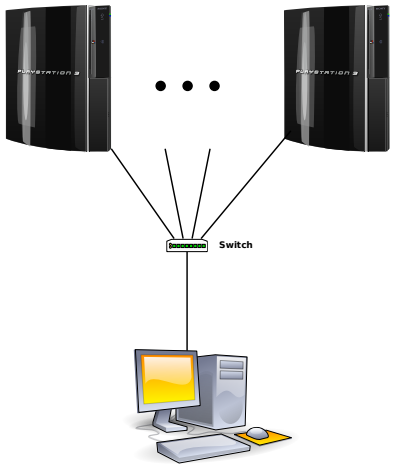

Figure 3. Cluster of PS3s and one workstation connected by a high speed gigabit switch.

6 Conclusion

Table 1 and Table 2, which show the wall time for the two image processing tasks that were implemented,

give a good feel at the performance that is available to the Cell BE. It is considerably faster than the Intel Pentium 4 processor and faster than the AMD Phenom 2 920 quad-core. The Phenom 2 became available in the market in early 2009 while the Cell BE could first be found in a commercial product in early 2005. So with these basic image processing tasks the Cell BE does better than a chip that is four years newer, which in terms of microprocessors is a very long time.

Implementing the two image processing tasks helped us learn some important things about programming on the cell processor.

1. Larger DMA transfers results in greater memory throughput but increases latency slightly.

2. The method by which memory is allocated in RAM can affect memory transfer significantly.

3. Double or multi-buffering is a simple but effective way to hide memory access.

4. In order to take advantage of autoSIMDization the way loops are organized can be very important.

5. The overhead of launching threads to run on the SPEs is expensive and should be avoided if possible.

The PS3 does have a few shortcomings that will limit what applications can be implemented. The biggest shortcomings are the lack of main memory, with only 256 MB, and the limited size of the LS. Some application performance will suffer severely because of the lack of space.

Thus far we do not have any performance numbers for the cluster of PS3s, but the information gathered so far indicates that a cluster of PS3 could provide very good performance for medical image processing. Most of the applications that would run on the PS3 are embarrassingly parallel and so the performance of the cluster should scale linearly with the number of nodes in the cluster. Given the price of a PS3, building a cluster of them is a very cheap method to get high performance results.

Acknowledgement

This work research was supported in part by the National Library of Medicine (NLM) under Grant contract HHSN 276200800412 and intramural research funds of the Lister Hill National Center for Biomedical Communications, the National Library of Medicine (NLM), and the National Institutes of Health (NIH).

References

[1] IBM. The Cell Project at IBM Research, September 2008. `http://www.research.ibm.com/cell/home.html`.

[2] Top500 Supercomputing Sites, November 2008. `http://www.top500.org/lists/2008/11`.

[3] The Green500 List, November 2008. `http://green500.org/lists/2008/11/list.php`.

[4] Joseph D. Dumas II. *Computer architecture: fundamentals and principles of computer design*, chapter six, pages 249–281. CRC Press, 2006.

[5] C. Reynolds. Big fast crowds on PS3. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 113–121. ACM New York, NY, USA, 2006.

[6] J. Kurzak, A. Buttari, P. Luszczek, and J. Dongarra. The PlayStation 3 for High-Performance Scientific Computing. *Computing in Science & Engineering*, 10(3):84–87, 2008.

[7] A. Buttari, P. Luszczek, J. Kurzak, J. Dongarra, and G. Bosilca. A rough guide to scientific computing on the PlayStation 3. *Innovative Computing Laboratory, Computer Science Department, University of Tennessee*, 2007.

[8] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The potential of the cell processor for scientific computing. In *Proceedings of the 3rd conference on Computing frontiers*, pages 9–20. ACM New York, NY, USA, 2006.

[9] *Cell Broadband Engine Programming Handbook version 1.1.*

[10] A.R. Smith. Color gamut transform pairs. *ACM SIGGRAPH Computer Graphics*, 12(3):12–19, 1978.

[11] J.G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A*, 2(7):1160–1169, 1985.

[12] S. Marcelja. Mathematical Description of the Responses of Simple Cortical Cells. *Journal of the Optical Society of America*, 70(11):1297–1300, 1980.

[13] Open MPI: Open Source High Performance Computing. `http://www.open-mpi.org/`.